BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

End of Result Set

L18: Entry 7 of 7 File: USPT Aug 21, 2001

US-PAT-NO: 6279100
DOCUMENT-IDENTIFIER: US 6279100 B1

TITLE: Local stall control method and structure in a microprocessor

DATE-ISSUED: August 21, 2001

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Tremblay; Marc	Menlo Park	CA		
Yeluri; Sharada	Sunnyvale	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Sun Microsystems, Inc.	Palo Alto	CA			02

APPL-NO: 09/ 204535 [PALM]
DATE FILED: December 3, 1998

PARENT-CASE:
CROSS-REFERENCE The present invention is related to subject matter disclosed in the following patent applications: 1. U.S. patent application Ser. No. 09/204,480 entitled, "A Multiple-Thread Processor for Threaded Software Applications", naming Marc Tremblay and William Joy as inventors and filed on even date herewith; 2. U.S. patent application Ser. No. 09/204,584 entitled, "Clustered Architecture in a VLIW Processor", naming Marc Tremblay and William Joy as inventors and filed on even date herewith; 3. U.S. patent application Ser. No. 09/204,481 entitled, "Apparatus and Method for Optimizing Die Utilization and Speed Performance by Register File Splitting", naming Marc Tremblay and William Joy as inventors and filed on even date herewith; 4. U.S. patent application Ser. No. 09/204,536 entitled, "Variable Issue-Width VLIW Processor", naming Marc Tremblay as inventor and filed on even date herewith; 5. U.S. patent application Ser. No. 09/204,586, now U.S. Pat. No. 6,205,543, entitled, "Efficient Handling of a Large Register File for Context Switching", naming Marc Tremblay and William Joy as inventors and filed on even date herewith; 6. U.S. patent application Ser. No. 09/205,121 entitled, "Dual In-line Buffers for an Instruction Fetch Unit", naming Marc Tremblay and Graham Murphy as inventors and filed on even date herewith; 7. U.S. patent application Ser. No. 09/204,781 entitled, "An Instruction Fetch Unit Aligner", naming Marc Tremblay and Graham Murphy as inventors and filed on even date herewith; 8. U.S. patent application Ser. No. 09/204,585 entitled, "Local and Global Register Partitioning in a VLIW Processor", naming Marc Tremblay and William Joy as inventors and filed on even data herewith; and 9. U.S. patent application Ser. No. 09/204,479 entitled, "Implicitly Derived Register Specifiers in a Processor", naming Marc Tremblay and William Joy as inventors and filed on even data herewith.

INT-CL: [07] G06 F 9/38

US-CL-ISSUED: 712/24; 712/219
US-CL-CURRENT: 712/24; 712/219

FIELD-OF-SEARCH: 712/24, 712/219

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected | Search ALL | Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	5222240	June 1993	Patel	712/218
☐	5524263	June 1996	Griffith	712/23
☐	5537561	July 1996	Nakajima	395/375
☐	5657291	August 1997	Podlesny et al.	365/230.05
☐	5721868	February 1998	Yung et al.	711/149
☐	5761475	June 1998	Yung et al.	712/218
☐	5764943	July 1998	Wechsler	712/218
☐	5778248	July 1998	Leung	712/23
☐	5787303	July 1998	Ishikawa	395/800.24
☐	6055620	April 2000	Paver	712/201

FOREIGN PATENT DOCUMENTS

FOREIGN-PAT-NO	PUBN-DATE	COUNTRY	US-CL
0 649 085	April 1985	EP	
0 730 223	September 1996	EP	
WO 96/27833	September 1996	WO	

OTHER PUBLICATIONS

Kazuaki Murakami et al: "SIMP (Single Instruction stream/Multiple instruction Pipelining): A Novel High-Speed Single-Processor Architecture" Computer Architecture News, US, Association for Computing Machinery, New York, vol. 17, No. 3, June 1, 1989, pp. 78-85, XP000035291 ISSN: 0163-5964.

ART-UNIT: 213

PRIMARY-EXAMINER: Coleman; Eric

ATTY-AGENT-FIRM: Skjerven Morrill MacPherson LLP Koestner; Ken J.

ABSTRACT:

A processor implements a local stall functionality in which small, independent circuit units are stalled locally with the condition causing a stall being first detected locally, then propagated to other small independent circuit units. Stall conditions for a functional unit are detected locally with reduced logic circuitry and also without waiting to receive condition information from other functional units that is transmitted over long wires. Local stall logic circuits are

distributed over diverse areas of an integrated circuit so that stall conditions are detected locally. A local stall is expanded into a global stall by propagation to logic circuits beyond a local region in subsequent cycles. Local detection of stall conditions and local stalling eliminates many critical paths in the processor.

27 Claims, 42 Drawing figures

Previous Doc Next Doc Go to Doc#

L18: Entry 7 of 7 File: USPT Aug 21, 2001

DOCUMENT-IDENTIFIER: US 6279100 B1
TITLE: Local stall control method and structure in a microprocessor

Abstract Text (1):
A processor implements a local stall functionality in which small, independent circuit units are stalled locally with the condition causing a stall being first detected locally, then propagated to other small independent circuit units. Stall conditions for a functional unit are detected locally with reduced logic circuitry and also without waiting to receive condition information from other functional units that is transmitted over long wires. Local stall logic circuits are distributed over diverse areas of an integrated circuit so that stall conditions are detected locally. A local stall is expanded into a global stall by propagation to logic circuits beyond a local region in subsequent cycles. Local detection of stall conditions and local stalling eliminates many critical paths in the processor.

INVENTOR (1):
Tremblay; Marc

Assignee Name (1):
Sun Microsystems, Inc.

Brief Summary Text (7):
The conditions occasionally necessitate stalling of the pipeline. A stall in a pipelined processor typically is handled by allowing some instructions to proceed while other instructions are delayed. The typical response when an instruction is stalled is to also stall all instructions that are subsequent to the stalled instruction in the pipeline. Instructions that are earlier in the pipeline are not stalled but no new instructions are fetched during the stall.

Brief Summary Text (9):
The control logic typically attempts to stall the front end of the pipeline. In some processors, the control logic attempts to stall the entire pipeline. Since the various stages of the pipeline are associated with all aspects of instruction functionality, including instruction fetching, decoding, execution of an entire range of instructions, trapping, writeback, and the like, stall signals are routed across essentially the entire functional layout of the integrated circuit. Therefore, the stall signals are propagated along long wires having lengths over many millimeters. The stall signals are typically propagated the functionally long distances within a single clock cycle. Accordingly, the stall logic is often a critical timing path and cycle time limitation in high-performance processors.

Brief Summary Text (12):
A processor implements a local stall functionality in which small, independent circuit units are stalled locally with the condition causing a stall being first detected locally, then propagated to other small independent circuit units. Stall conditions for a functional unit is detected locally with reduced logic circuitry and also without waiting to receive condition information from other functional units that is transmitted over long wires. Local stall logic circuits are distributed over diverse areas of an integrated circuit so that stall conditions are detected locally. A local stall is expanded into a global stall by propagation to logic circuits beyond a local region in subsequent cycles. Local detection of stall conditions and local stalling eliminates many critical paths in the processor.

Drawing Description Text (11):
FIGS. 9A-9C respectively show an instruction sequence table, and two pipeline diagrams illustrating execution of a VLIW group which shows stall operation for a five-cycle pair instruction and a seven-cycle pair instruction.

Drawing Description Text (23):
FIGS. 21A, 21B, and 21C are, respectively, a VLIW packet diagram, a pipeline diagram, and a timing diagram which illustrate an operation of updating an E-stage scoreboard entry in the presence of a D-stage stall.

Detailed Description Text (17):
The processor 100 supports full bypasses between the first two execution units within the media processing unit 110 and has a scoreboard in the general functional unit 222 for load operations so that the compiler does not need to handle nondeterministic latencies due to cache misses. The processor 100 scoreboards long latency operations that are executed in the general functional unit 222, for example a reciprocal square-root operation, to simplify scheduling across execution units. The scoreboard (not shown) operates by tracking a record of an instruction packet or group from the time the instruction enters a functional unit until the instruction is finished and the result becomes available. A VLIW instruction packet contains one GFU instruction and from zero to three MFU instructions. The source and destination registers of all instructions in an incoming VLIW instruction packet are checked against the scoreboard. Any true dependencies or output dependencies stall the entire packet until the result is ready. Use of a scoreboarded result as an operand causes instruction issue to stall for a sufficient number of cycles to allow the result to become available. If the referencing instruction that provokes the stall executes on the general functional unit 222 or the first media functional unit 220, then the stall only endures until the result is available for intra-unit bypass. For the case of a load instruction that hits in the data cache 106, the stall may last only one cycle. If the referencing instruction is on the second or third media functional units 220, then the stall endures until the result reaches the writeback stage in the pipeline where the result is bypassed in transmission to the split register file 216.

Detailed Description Text (120):
The pcu/1su load/store buffer empty signal is asserted late in cycle 2 at the same time the load is sent to the load/store unit 218. The pipeline control unit 226 latches the signal and monitors the signal in cycle 3 to stall the membar instruction. The data cache hit data returns in cycle 3. The load/store unit 218 asserts the pcu/1su load/store buffer empty signal late in cycle 4. The pipeline control unit 226 asserts the stall (signal "mbar_staller") in cycle 5.

Detailed Description Text (143):
Referring to FIGS. 21A, 21B, and 21C respectively, a VLIW packet diagram, a pipeline diagram, and a timing diagram illustrate an operation of updating an E-stage scoreboard entry in the presence of a D-stage stall. If a load or long-latency instruction stalls in the D-stage, then the instruction should not update the scoreboard so the scoreboard will not self-interlock in the next clock cycle. Since the scoreboard write and reset pointers are generated in the general functional unit controller 1912 and routed to all scoreboards, qualifying the write pointer with a D-stage stall, which is generated in about 1.6 ns, may become timing critical. Therefore, in the illustrative implementation, a write to the scoreboard entry is not qualified by the D-stage stall in the general functional unit 222. Any hit to the esb_entry of the scoreboard in any cycle is qualified by the absence of the D-stage stall in the previous cycle.

CLAIMS:

1. A processor comprising:

an instruction buffer including a plurality of segments that store a plurality of subinstructions corresponding to respective segments of the plurality of segments;

a plurality of functional units coupled to the instruction buffer, the plurality of functional units that receive the corresponding plurality of subinstructions for execution; and

a pipeline control unit including a plurality of pipeline control segments respectively coupled to corresponding functional units of the plurality of functional units, the pipeline control segments controlling the pipeline of the corresponding functional unit to detect a stall condition that is local to a functional unit and stall the local functional unit independently of nonlocal functional units in an initial timing cycle, and propagating a stall to the nonlocal functional units in a subsequent timing cycle.

2. The processor according to claim 1 further comprising:

stall detection logic circuitry in a pipeline control unit that detects stall conditions only local to a corresponding functional unit without waiting to receive condition information from other functional units that is transmitted over long wires.

11. The processor according to claim 1 further comprising:

a scoreboard storage coupled to the pipeline control unit, the scoreboard storage including a plurality of fields that hold information relating to load operations and long-latency operations when operations transition from a decode stage to an execute stage, the scoreboard storage, the information including a field storing a destination register specifier of a load instruction or long-latency instruction, a field to designate whether an instruction is a load instruction or a long-latency instruction, and a field to indicate whether the instruction is a pair instruction, the pipeline control unit coupled to the scoreboard and initiating a stall according to relative positioning of load instructions, long latency instructions, and pair instructions in a pipeline.

12. The processor according to claim 1 further comprising:

a scoreboard storage coupled to the pipeline control unit, the scoreboard storage including a field that holds information relating to a CALL operations, the pipeline control unit coupled to the scoreboard and initiating a stall to prevent conflict between an unfinished load operation and a CALL return operation.

20. A processor comprising:

an instruction buffer including a plurality of segments that store a plurality of subinstructions corresponding to respective segments of the plurality of segments;

a plurality of functional units coupled to the instruction buffer, the plurality of functional units that receive the corresponding plurality of subinstructions for execution; and

a pipeline control unit including a plurality of pipeline control segments respectively coupled to corresponding functional units of the plurality of functional units, the pipeline control segments controlling the pipeline of the corresponding functional unit: (1) stalling the entire plurality of subinstructions of an instruction word for a true dependency or an output dependency, (2) stalling only until a result is available for intra-functional unit bypass for a stall condition in a first priority functional unit, and (3) stalling until a result reaches a pipeline writeback stage for a stall condition in a second priority functional unit.

21. A processor according to claim 20 wherein:

the plurality of pipeline control segments are local pipeline control segments that are respectively coupled to corresponding local functional units of the plurality of functional units,

the local pipeline control segments locally control the pipeline of the corresponding local functional unit independently of condition information of nonlocal functional units in a timing cycle that a stall condition is detected and propagate the detected stall condition to nonlocal functional units in subsequent cycles.

22. A processor according to claim 20 wherein:

the plurality of pipeline control segments are local pipeline control segments that are respectively coupled to corresponding local functional units of the plurality of functional units, and

a stall occurs locally within the local pipeline control segments independently of other pipeline control segments in a first timing cycle, the local stall expanding to the other pipeline control segments in subsequent timing cycles.

23. A method of executing instructions in a processor comprising:

storing an instruction word in an instruction buffer, the instruction buffer including a plurality of segments that store a respective plurality of subinstructions that, in combination, make up the instruction word;

receiving from the instruction buffer a plurality of subinstructions for execution at a plurality of functional units that are coupled to the instruction buffer; and

controlling a plurality of instruction pipelines, individual pipelines receiving a subinstruction from a corresponding instruction buffer segment and executing the subinstruction in a corresponding functional unit; and

controlling the pipeline of the corresponding functional unit to detect a stall condition that is local to a functional unit and stall the local functional unit independently of nonlocal functional units in an initial timing cycle, and propagating a stall to the nonlocal functional units in a subsequent timing cycle.

Previous Doc Next Doc Go to Doc#

L27: Entry 6 of 13 File: USPT Aug 19, 2003

US-PAT-NO: 6609193
DOCUMENT-IDENTIFIER: US 6609193 B1
**** See image for Certificate of Correction ****

TITLE: Method and apparatus for multi-thread pipelined instruction decoder

DATE-ISSUED: August 19, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Douglas; Jonathan P.	Portland	OR		
Deleganes; Daniel J.	Hillsboro	OR		
Hadley; James D.	Portland	OR		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Intel Corporation	Santa Clara	CA			02

APPL-NO: 09/ 475105 [PALM]
DATE FILED: December 30, 1999

INT-CL: [07] G06 F 9/38

US-CL-ISSUED: 712/219; 709/107, 712/208, 713/601
US-CL-CURRENT: 712/219; 712/208, 713/601, 718/107

FIELD-OF-SEARCH: 712/208, 712/219, 709/107, 713/601

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	5357617	October 1994	Davis et al.	712/245
☐	5778246	July 1998	Brennan	712/23
☐	5890008	March 1999	Panwar et al.	712/15
☐	5913049	June 1999	Shiell et al.	712/215
☐	5983339	November 1999	Klim	712/200
☐	6026476	February 2000	Rosen	711/206

☐	6357016	March 2002	Rodgers et al.	713/601
☐	6385719	May 2002	Derrick et al.	712/235

FOREIGN PATENT DOCUMENTS

FOREIGN-PAT-NO	PUBN-DATE	COUNTRY	US-CL
WO93/01545	July 1992	WO	

OTHER PUBLICATIONS

Eugene R. Hnatek; Random-Access Memories and Content-Addressable Memories, A User's Handbook of Semiconductor Memories; 1977; pp. 591-608; Wiley-Interscience Pub.
William Stallings; 7.3 Memory Management, Computer Organization and Architecture, Desigining for Performance, 4th Edition; 1996; pp. 240-263; Prentice Hall, New Jersey.
Hamcher, Vranesic & Zaky; The Main Memory, Computer Organization, 2nd Edition; 1984; pp. 306-329; McGraw-Hill Book Company.
Patterson & Hennessy; Memory-Hierachy Design, Computer Architecture: A Quantitative Approach; 1990; pp. 408-475; Morgan Kaufmann Publishers, San Mateo, CA.
Richard Kain; Advanced Computer Architecture: A Systems Design Approach; 1996; pp. 75-88 and 456-474; Prentice Hall, Englewood Cliffs, New Jersey.
Jean-Loup Baer; Computer Systems Architecture; 1980; pp. 139-166 and 315-325; Computer Science Press, Rockville, MD.

ART-UNIT: 2183

PRIMARY-EXAMINER: Kim; Kenneth S.

ATTY-AGENT-FIRM: Blakely, Sokoloff, Taylor & Zafman LLP

ABSTRACT:

A multithread pipelined instruction decoder to clock, clear and stall an instruction decode pipeline of a multi-threaded machine to maximize performance and minimize power. A shadow pipeline shadows the instruction decode pipeline maintaining a the thread-identification and instruction-valid bits for each pipestage of the instruction decoder. The thread-id and valid bits are used to control the clear, clock, and stall of each pipestage of the instruction decoder. Instructions of one thread can be cleared without impacting instructions of another thread in the decode pipeline. In some cases, instructions of one thread can be stalled without impacting instructions of another thread in the decode pipeline. In the present invention, pipestages are clocked only when a valid instruction needs to advance in order to conserve power and to minimize stalling.

36 Claims, 12 Drawing figures

Previous Doc Next Doc Go to Doc#

DOCUMENT-IDENTIFIER: US 6609193 B1
**** See image for Certificate of Correction ****
TITLE: Method and apparatus for multi-thread pipelined instruction decoder

Detailed Description Text (22):
The control algorithm implemented by the control logic 401 of the present invention to support multi-threading in the pipeline instruction decoder 400' has three main functional parts: (1) Efficient Stalling and Bubble Squeezing, (2) Thread Specific Clearing, and (3) Opportunistic Powerdown. Referring now to FIGS. 6 and 7, FIG. 7 illustrates control algorithm equations executed by the control logic 401 of the present invention illustrated in FIG. 6. The power down logic 603 illustrated in FIG. 6, executes the "Powerdown for any PipeStage X" equation for each pipestage. In order to do so, the powerdown logic 603 has input the instruction valid bit of each pipestage. Additionally, the powerdown logic 603 executes the "Stall for Next to Last PipeStage (NLP)" equation and the "Stall for any other PipeStage (X)" equation illustrated in FIG. 7. In order to do so, the powerdown logic 603 additionally receives a thread stall signal with the thread-ID of the stall to determine if the next to last pipestage of the instruction decode pipeline should be stalled. The powerdown logic 603 processes the stall condition for each pipestage by ANDing the instruction valid bits of a given pipestage with the instruction valid bits of the subsequent pipestage and further ANDing these results with the determination of whether the next to last pipestage is stalled. The powerdown logic passes the stall condition for each stage to the clock control logic 604. The clock control logic selectively runs and stops the clock to each pipestage in accordance with the equation for "Clock Enable for any PipeStage X" illustrated in FIG. 7. If a given pipestage is not stalled and it is not powerdown, then the given pipestage has its clock enabled so that it can be clocked on the next cycle.

Detailed Description Text (24):
Examples of the functionality of the Efficient Stalling and Bubble Squeezing, Thread Specific Clearing, and Opportunistic Powerdown algorithms are now described with reference to FIGS. 8-10, 11A and 11B. The illustrations provided in FIGS. 8-10, 11A and 11B are associated with the control of the instruction decode pipeline 400' between buffer 502A and buffer 502B in FIG. 5. Pipestages 513 through 517 are referred to as pipestages PS1 through PS5 in the discussion below but can be generalized to the control of any instruction decode pipeline within an instruction decoder using the algorithms of the present invention.

Detailed Description Text (25):
Efficient Stalling and Bubble Squeezing

Detailed Description Text (27):
A bubble is a number of invalid instructions located within the instruction decoder. Usually the bubble is created as a result of an entire thread of instructions mixed amongst other instruction threads in the instruction decode pipeline becoming invalid. An example that would cause this is a misconnected branch. The bubble squeeze algorithm performed by the present invention generally squeezes out the bubbles of instructions in the instruction decode pipeline. The bubble squeeze algorithm is essentially accomplished by continuing to clock pipestages which have their instructions marked as invalid until a valid instruction is received. The clocks to a pipestage containing a valid instruction are temporarily stopped until the reason for the stall is cleared. The invalid instructions are eventually squeezed out by writing valid instructions over the invalid instructions stored in the pipestages. The bubble squeeze algorithm continues to run the instruction decode pipeline to bring instructions of other threads further down the pipeline instead of performing a non-intelligent or blocking stall. Bubble squeezing can provide greater

throughput in the instruction decoder.

Detailed Description Text (28):
This algorithm for efficient stalling and bubble squeezing processes the thread specific stalls including those generated by the variable consumption buffers. By using the thread-ID from the thread-ID pipeline and instruction valid bits of the instruction valid pipeline, the algorithm determines if a valid instruction of the thread-ID corresponding to the stall would be presented to the buffer in the next cycle. If so, then the next to last pipestage prior to the buffer is stalled (prevented from issuing any more instructions). The next to last pipestage is used instead of the last pipestage in order to provide a cycle time of evaluation in the preferred embodiment. In alternate embodiments, the last pipestage may be substituted for the next to last pipestage. Any other instruction decode pipestages that do not have a valid instruction are not stalled. Any instruction pipestages after the buffer are also not stalled. This allows instructions in the pipe to advance until the pipe is full, while still stalling the next to last pipestage to prevent an instruction from being lost, increasing overall decode bandwidth. If the instruction data about to enter the buffer is not of the same thread as the stall, then the clocks are kept running. This keeps instructions of another thread from being stalled and allows instructions of the same thread further back in the instruction decode pipeline to advance, thereby further increasing the bandwidth of the instruction decoder.

Detailed Description Text (61):
The algorithms for Efficient Stalling and Bubble Squeezing, Thread Specific Clearing, and Opportunistic Powerdown are inter-related. For example clearing a specific pipestage using a thread specific clear can cause a stall to be eliminated for a given pipestage. Alternatively, a thread specific clear may invalidate instructions in certain pipestages to provide an opportunistic powerdown condition.

Current US Original Classification (1):
712/219

Previous Doc Next Doc Go to Doc#

First Hit Fwd Refs Previous Doc Next Doc Go to Doc#

Generate Collection Print

L27: Entry 12 of 13 File: USPT Jun 28, 1994

US-PAT-NO: 5325495
DOCUMENT-IDENTIFIER: US 5325495 A

TITLE: Reducing stall delay in pipelined computer system using queue between pipeline stages

DATE-ISSUED: June 28, 1994

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
McLellan; Edward J.	Milford	MA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Digital Equipment Corporation	Maynard	MA			02

APPL-NO: 08/ 103815 [PALM]
DATE FILED: August 9, 1993

PARENT-CASE:
RELATED APPLICATIONS This application is a continuation of an earlier filed U.S. application Ser. No. 07/723,210, filed Jun. 28, 1991 and now abandoned.

INT-CL: [05] G06F 9/38

US-CL-ISSUED: 395/375; 364/DIG.1, 364/261.5, 364/263.1, 364/238.6, 364/262.4, 364/231.8, 395/800
US-CL-CURRENT: 712/219; 712/229

FIELD-OF-SEARCH: 364/DIG.1MSFile, 364/DIG.2MSFile, 395/375, 395/800

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4777594	October 1988	Jones et al.	364/200
☐	4974154	November 1990	Matsuo	395/375
☐	5019967	May 1991	Wheeler et al.	395/775

OTHER PUBLICATIONS

Troiani et al, "The VAX 8600 I Box, A Pipelined Implementation of the VAX Architecture", Digital

Technical Journal, Aug. 1985, pp. 24-42.
Kane, "MIPS R2000 RISC Architecture", Prentice Hall, 1987, pp. 1-1 to 1-14, 2-1 to 2-5, 2-12 to 2-13, 3-13 to 3-16.
Smith, "A Study of Branch Prediction Strategies", 8th Annual Symposium on Computer Architecture, May 12-14, 1981, pp. 135-148.
Bakoglu et al., "The IBM RISC System/6000 processor: Hardware Overview", IBM J. Res. Develop., Jan. 1990, pp. 12-22.
Oehler et al, "IBM RISC System/6000 processor architecture", IBM J. Res. Develop., Jan. 1990, pp. 23-36.
Grohoski, "Machine Organization of the IBM RISC System/6000 processor", IBM J. Res. Develop., Jan. 1990, pp. 37-58.
Losq, "Generalized History Table for Branch Prediction", IBM Tech. Discl. Bulletin, Jun. 1982, pp. 99-101.
Rao, "Techniques for Minimizing Branch Delay . . .", IBM Tech. Discl. Bulletin, Jun. 1982, pp. 97-98.

ART-UNIT: 235

PRIMARY-EXAMINER: Harrell; Robert B.

ATTY-AGENT-FIRM: Arnold, White & Durkee

ABSTRACT:

A pipelined computer system employs a queue stage to receive the output of one pipeline stage when a stall occurs in the next stage or downstream of the next stage. This avoids stalling earlier stages of the pipeline. Subsequently, the pipeline advances through the queue, until a bubble occurs. When a bubble is subsequently generated upstream and enters the queue stage, a multiplexer switches the input of the next stage to receive the output of the one stage instead of from the queue stage, and the content of the queue is overwritten. By this mechanism, the delays inherent in processing branches can be reduced.

20 Claims, 4 Drawing figures

Previous Doc Next Doc Go to Doc#

L27: Entry 12 of 13 File: USPT Jun 28, 1994

DOCUMENT-IDENTIFIER: US 5325495 A
TITLE: Reducing stall delay in pipelined computer system using queue between pipeline stages

Brief Summary Text (8):
In U.S. Pat. No. 5,019,967, issued to William R. Wheeler and George M. Uhler, assigned to Digital Equipment Corporation, a method of pipeline bubble compression in a computer system is disclosed. This method provides a way of compressing bubbles by overwriting a bubble when a stall condition is detected downstream of the bubble. This may be referred to as a bubble squash technique.

Brief Summary Text (11):
If it was known that the assembly line never quite reached it's peak operating efficiency due to occasional parts missing in forward assembly stages, then an opportunity would exist to combine the two negative effects and effectively mask out the problem with red car drying time. Assume that each assembly stage were allowed to accept new work whenever they had a free time-slot. The red car pipeline bubble could then be eliminated at the time of the first missing part pipeline stall. The stage that was idle in the last time-slot due to the red car bubble, would accept a car as would all stages feeding it. All later stages would be forced to stall while waiting for the missing part before advancing because their respective receiving stages would still be occupied by cars that could not advance. This situation is an improvement over the original pipeline sequence because the red car bubble would be effectively eliminated by combining the two potentially negative events. The drawback is that it may not be practical for half of the assembly line to proceed while the other half is stalled if, for instance, all cars advance on one large treadmill. In addition, if the parts supplier suddenly improves his performance, thus eliminating most of the missing part stalls, then the opportunity to mask the red car bubble would be dramatically reduced and the expected improvement provided by the now ample parts supply may not be realized. More and more of the red car bubbles would progress through the entire pipeline and contribute toward reduced production. This scenario is similar to the bubble squash algorithm implemented in the previous designs. As long as additional stalls occur while branch delay bubbles exist in specific pipeline stages, they can be combined to mask the branch delay performance impact.

Brief Summary Text (12):
If the parts supplier improved his performance greatly but still encountered trouble at times, in other words, the pipeline encounters stalls occasionally, but much less frequently, then it would be an improvement to "save" the occasional stall cycle for later combination with a red car bubble. A method of accomplishing this would be to add a queue stage off to the side of the pipeline. The pipeline would function normally until there was a missing part stall. At this time, the stage immediately preceding and feeding the missing part stage would place it's car in the queue stage and accept new work. All stages behind could advance as if there were no stall in that time-slot. In this case, we assume that the queue is only able to hold one car at a time. When the missing part is delivered, the pipeline is allowed to proceed with the queue stage now incorporated as an integral part of the pipeline. It accepts a car each time-slot and advances a car to the missing part stage each time-slot. It does not otherwise accomplish any work. When a red car bubble advances to the newly inserted queue stage, the queue no longer is needed because it does not hold a car. When the time-slot expires, the missing part stage accepts a car directly from it's preceding stage and not the queue. In this way, the red car bubble has been effectively eliminated from the pipeline without requiring that the missing part stall occur during the time the bubble was traveling through the line. Instead, a missing part stall that occurs at any time after the last red car advances to the missing part stage, but before a new red car bubble reaches the missing part stage, is sufficient to eliminate the red car bubble. The advantage of this scheme is directly related to the relaxation of the timing requirements of missing part stalls

compared to red car bubbles. In a computer system, this equates to the relationship between the occurrence of pipeline stalls relative to the occurrence of taken branch delay slots.

Brief Summary Text (13):
Compared with the bubble squash algorithm, an additional advantage is gained by avoiding the complication of multiple pipeline advance commands necessary to allow bubble squashing at multiple pipeline stages. Under the bubble squash scheme, each stage capable of advancing while the pipeline was otherwise stalled required a separate control equation. This is required because the ability to advance a stage while a stall condition exists implies knowledge of data in the pipeline. A pipeline stage can advance in the presence of a stall if the stage that it feeds contains a bubble or if any of the stages ahead of that stage are advancing. If multiple stages are used to trap bubbles in an effort to maximize the probability of a squash, then the equations become supersets of one another.

Detailed Description Text (2):
Referring to FIG. 1, stages of a pipelined computer system are illustrated according to one embodiment of the invention. The computer itself may be, for example, a 64-bit RISC architecture as described in copending application Ser. No. 547,630, filed Jun. 29, 1990, by Richard L. Sites and Richard T. Witek, assigned to Digital Equipment Corporation. The computer system includes successive operational stages 10, 11, 12, and 13 which may be performing functions such as instruction fetch, operand fetch, arithmetic/logic functions, etc. Control stages #1, #2, #3, and #4 temporarily hold sets of the control bits produced by an instruction decoder 14, and these sets are clocked through the control stages #1-#4 on successive clock cycles in a pipelined manner. The operational stages 10, 11, 12 and 13 are in lock step with the control stages, so each operational stage and control stage together define a pipeline stage. Referring to FIG. 2, assuming no stalls or bubbles, control bits for instruction-A would enter stage #1 at cycle-1, then proceed through stages #2, #3 and #4 in cycle-2 to cycle-4. Meanwhile, control bits for instruction-B enter stage #1 at cycle-2, instruction-C at cycle-3, etc. In cycle-4, there are four instructions in the pipeline, a different one in each of the stage #1-#4. While in a stage, the sets of control bits are applied by lines 15 to the controlled operational stages. There may be additional stages to the pipeline, of course. The RISC processor disclosed in the above-mentioned application Ser. No. 547,630 has a seven stage pipeline.

Detailed Description Text (5):
The queue stage 16 normally loads via output 17 with the same data as stage #4. When a pipeline stall occurs in stage #4 or downstream, the advance command on control lines 19 from pipeline control 20 to stages #1, #2, #3 and the queue stage 16 is asserted even though stage #4 cannot accept new input. Stage #3 sends it's output on lines 17 to the queue stage 16 to avoid stalling the earlier pipeline stages. In addition, this action arms the queue mechanism to catch and trap bubbles. When a bubble enters the queue stage 16 (or the queue stage is empty), the multiplexer 18 switches back to accepting output from stage #3, thus allowing the pipeline to go back to being a four stage pipeline instead of a five stage pipeline by omitting the queue stage. When this switch is made, the bubble may be considered to be overwritten in the queue stage 16, though this is not necessary to eliminate the bubble (it would be overwritten, in any event, upon the next stall condition).

Detailed Description Text (14):
Relocating the queue stage 16 is an implementation option; however, any movement of the queue state 16 toward the beginning of the pipeline and away from the source of stalls creates the opportunity for bubbles to advance past the queue stage 16 before a stall cycle occurs. If a bubble advances past the queue stage 16, all opportunity to squash it using the queue has been lost. In this case, a combination of queue stage and other styles of bubble squash hardware can be employed to recoup the potential loss of performance. This trade-off is dependent on the distribution of stall cycles relative to causes of pipeline bubbles. If stall cycles typically occur while processing the instructions closely preceding branch instructions, but not often at other times, then moving the queue 16 can decrease the probability of catching a bubble. If, on the other hand, stall cycles occur often, or with high probability in cycles preceding a branch instruction but not closely preceding, then the probability of catching and squashing the branch delay slot bubble may not be adversely affected.

Current US Original Classification (1):
712/219

CLAIMS:

1. A pipelined computer system, comprising:

a plurality of pipeline stages, each said pipeline stage having an input and an output, said input of each of said pipeline stages receiving information from said output of a preceding one of said pipeline stages in a normal mode of operation;

a bypass queue connected to receive said output of one of said pipeline stages, and switch means to route said output of said one of said pipeline stages through said bypass queue instead of to said input of a next one of said pipeline stages and to connect an output of said bypass queue to said input of said next one of said pipeline stages, in a bypass mode initiated in response to a command;

control means responsive to a stall occurring in said pipeline downstream of said one of said stages, to generate said command and initiate said bypass mode, said bypass mode being maintained until a bubble enters said bypass queue;

said control means being responsive to a bubble entering said queue stage to terminate said bypass mode and initiate said normal mode and cause said next one of said stages to receive the output of said one stage directly via said switch means.

Previous Doc Next Doc Go to Doc#

First Hit Fwd Refs Previous Doc Next Doc Go to Doc#

End of Result Set

Generate Collection Print

L27: Entry 13 of 13 File: USPT May 28, 1991

US-PAT-NO: 5019967
DOCUMENT-IDENTIFIER: US 5019967 A
**** See image for Certificate of Correction ****

TITLE: Pipeline bubble compression in a computer system

DATE-ISSUED: May 28, 1991

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Wheeler; William R.	Hudson	MA		
Uhler; George M.	Marlborough	MA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Digital Equipment Corporation	Maynard	MA			02

APPL-NO: 07/ 221988 [PALM]
DATE FILED: July 20, 1988

INT-CL: [05] G06F 9/38

US-CL-ISSUED: 364/200; 364/263, 364/231.8, 364/262.4, 364/271.5
US-CL-CURRENT: 712/219

FIELD-OF-SEARCH: 364/2MSFile, 364/9MSFile

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	3248707	April 1966	Paul et al.	364/200
☐	4090249	May 1978	Chen et al.	364/900
☐	4236206	November 1980	Stecker et al.	364/200
☐	4438492	March 1984	Harmon, Jr. et al.	364/200
☐	4498136	February 1985	Sproul, III	364/200
☐	4534009	August 1985	McGee	364/900

☐	4541047	September 1985	Wada et al.	364/200
☐	4586130	April 1986	Butts, Jr. et al.	364/200
☐	4594655	June 1986	Hao et al.	364/900
☐	4636943	January 1987	Horst et al.	364/200
☐	4670836	June 1987	Kibo et al.	364/200
☐	4701842	October 1987	Olnawich	364/200
☐	4701915	October 1987	Kitamura et al.	364/200
☐	4709324	November 1987	Kloker	364/200
☐	4750112	June 1988	Jones et al.	364/200
☐	4760519	July 1988	Papworth et al.	364/200
☐	4794517	December 1988	Jones et al.	364/200
☐	4794518	December 1988	Mizushima	364/200
☐	4794527	December 1988	Stewart et al.	364/200

OTHER PUBLICATIONS

Mishra, "The VAX 8800 Architecture", Digital Technical Journal, Feb. 1987, pp. 20-33.
Troiani et al., "the VAX 8600 I Box, A Pipelined Implementation of the VAX Architecture", Digital Technical Journal, Aug. 1985, pp. 24-42.

ART-UNIT: 237

PRIMARY-EXAMINER: Shaw; Gareth D.

ASSISTANT-EXAMINER: Kriess; Kevin A.

ATTY-AGENT-FIRM: Arnold, White & Durkee

ABSTRACT:

Bubble compression in a pipelined central processing unit (CPU) of a computer system is provided. A bubble represents a stage in the pipeline that cannot perform any useful work due to the lack of data from an earlier pipeline stage. When a particular pipeline stage has stalled, the CPU instructions that have already passed through the stage continue to move ahead and leave behind vacant stages or bubbles. If a bubble is introduced into a pipeline and the pipeline subsequently stalls, the disclosed CPU takes advantage of this stalled condition to compress the previously introduced bubble.

21 Claims, 11 Drawing figures

Previous Doc Next Doc Go to Doc#

L27: Entry 13 of 13 File: USPT May 28, 1991

DOCUMENT-IDENTIFIER: US 5019967 A
**** See image for Certificate of Correction ****
TITLE: Pipeline bubble compression in a computer system

Abstract Text (1):
Bubble compression in a pipelined central processing unit (CPU) of a computer system is provided. A bubble represents a stage in the pipeline that cannot perform any useful work due to the lack of data from an earlier pipeline stage. When a particular pipeline stage has stalled, the CPU instructions that have already passed through the stage continue to move ahead and leave behind vacant stages or bubbles. If a bubble is introduced into a pipeline and the pipeline subsequently stalls, the disclosed CPU takes advantage of this stalled condition to compress the previously introduced bubble.

Brief Summary Text (34):
Stalling introduces bubbles in the pipeline. A bubble represents a stage in the pipeline that cannot accomplish any useful work due to the lack of data from an earlier pipeline stage. As a bubble propagates through the pipeline it causes the corresponding functional units to become idle. In effect, a pipeline bubble is a lost opportunity to do useful work and results in lower processor throughput. This invention deals with a CPU pipeline implementation that compresses bubbles.

Brief Summary Text (39):
This invention allows pipeline stages of a digital computer CPU to be advanced in a manner that compresses bubbles. A bubble represents a stage in the pipeline that cannot perform any useful work due to the lack of data from an earlier pipeline stage. When a particular pipeline stage stalls, the CPU instructions that have already passed through this stage continue to move ahead and leave behind vacant stages or bubbles. Those vacant stages do not have new instructions to process because of the stall.

Brief Summary Text (40):
Once a bubble is introduced into the pipeline, it propagates through because, unless there is a stall, all stages are advanced simultaneously. In previous designs, if a stall occurred, all stages prior to the stalled stage also halted and any bubbles in this region stayed as they were. This invention causes any bubbles prior to the stalled stage to be filled in, by allowing the stages prior to the stalled stage to move if bubbles are detected prior to the stalled stage.

Brief Summary Text (41):
In an embodiment, this invention is implemented by controlling the advance of pipeline stages prior to a bubble if one of the stages after the bubble has stalled. This involves detection of bubbles and stalls in each stage, comparing their relative positions and conditionally advancing each pipeline stage.

Detailed Description Text (63):
In cycle-5, the microsequencer 23 gets a stall dispatch (bubble) from the latch 102. This causes the microsequencer 23 to fetch another end-flow instruction and thus continue to assert a signal on the decoder-next line. The latch 105 loads in the dispatch address of the micro-flow starting with the microword Q from segment-1. Segment-1 produces a new dispatch address for a micro-flow starting with the microword T.

Detailed Description Text (64):
In cycle-6, the microsequencer 23 gets the dispatch address (address for microword Q) from the latch 101. This latch gets a new dispatch address (address for microword T) from segment-1. Segment-1 produces a new dispatch address for a micro-flow starting with the microword Y. In this way, the bubble created by the first pipeline stall is propagated through the pipeline segments.

Detailed Description Text (66):
Cycle 0, 1 and 2 are essentially the same as in Table I. The only difference is that an overwrite flag is now set in the latches 102, 105, 107, 108 when these contain a stall dispatch in cycle-1 and cycle-2; this informs segment-1 that these latchs contain a bubble and can be overwritten during the next cycle regardless of whether the pipeline is stalled.

Detailed Description Text (68):
In cycle-4, the microsequencer 23 reaches the end of the micro-flow and requests a new micro-flow dispatch address, by asserting a signal on the decoder-next line. Because of the overwriting of the latches 102, etc., in cycle-3, the microsequencer 23 now gets a valid dispatch address instead of a stall dispatch which would have resulted if bubble compression was not done.

Detailed Description Paragraph Table (1):
TABLE I MICROWORD
ADDRESSED CYCLE OUTPUT FROM BY MICROSEQUENCER NUMBER SEGMENT-1 (SEGMENT-2 INPUT) SEGMENT-3
COMMENTS 0 Stall dispatch Dispatch address for "end-flow" micro Decoder.sub.-- next set causes pipeline microword B instruction; to advance next cycle Decoder.sub.-- next set 1 Stall dispatch Stall dispatch: Microword B Stall dispatch loaded in the (bubble) Latch 102. Bubble created there. 2 Decode CPU Stall dispatch: Microword C Segment-1 produces valid dispatch Instruction to (bubble) address. segment-2 stalled. produce dispatch address for microword Q 3 Decode CPU Stall dispatch: Microword D Bubble still present in Instruction to (bubble) Latch 102. Microsequencer produce dispatch continues fetching microinstruc- address for tions of micro-flow. microword Q 4 Decode CPU Stall dispatch: "end-flow" micro Decoder.sub.-- next Set causes pipeline Instruction to (bubble) instruction; to advance in next cycle. produce dispatch Decoder.sub.-- next address for set microword Q 5 Decode CPU Dispatch for " end-flow" micro Stall dispatch causes "end-flow" Instruction to microword Q instruction; micro instruction to be fetched again. produce dispatch Decoder.sub.-- next Decoder.sub.-- next set again. address for set again microword T 6 Decode CPU Dispatch for Microword Q Pipeline continues to advance in Instruction to microword T normal manner. produce dispatch address for microword Y

Current US Original Classification (1):
712/219

CLAIMS:

1. A method of operating a pipelined processing unit in a digital computer, said pipelined processing unit having at least a first pipeline segment and a second pipeline segment for processing information, said first pipeline segment processing information in said pipeline upstream of said second pipeline segment; said processing of information in said second pipeline segment being capable of causing a stall condition which results in waiting for processing information and capable of producing a bubble in said first pipeline segment during which said first pipeline segment does no useful processing of information; and control means in said pipelined processing unit for controlling said processing of information and responsive to said stall condition and said bubble; said method comprising the steps of:

a) detecting a bubble in the first pipeline segment by said control means;

b) detecting a stall condition in the second pipeline segment by said control means; and

c) overwriting the bubble in the first pipeline segment by said processing of information, under control by said control means, thereby compressing the bubble.

19. A method of operating a pipelined processor having a plurality of pipelined stages, where each stage receives input data from a preceding stage and passes output data to a succeeding stage, and where one of said stages can send a stall control to a preceding stage upstream of said one stage, and wherein said preceding stage can operate in a bubble condition of passing useless output data to a succeeding stage, comprising the steps of:

(a) detecting said stall control from said one stage of said pipeline stages;

(b) determining that the output data in said preceding stage represents useless output data; and

(c) overwriting the output data of said preceding stage with input data received by said preceding stage.

Previous Doc Next Doc Go to Doc#

WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Sunday, July 11, 2004

Hide?	Set Name	Query	Hit Count
	DB=PGPB,USPT; PLUR=NO; OP=ADJ		
☐	L27	L25 and L26	13
☐	L26	(stall$3 with bubble$1)	192
☐	L25	(712/219).ccls.	224
☐	L24	(L22 or L23) and bubble$1	1
☐	L23	6279100.pn.	1
☐	L22	6044456.pn.	1
☐	L21	L18 and delay$3	7
☐	L20	6044456.uref.	2
☐	L19	6279100.uref.	6
☐	L18	global stall and L16	7
☐	L17	global stall and L5L5	0
☐	L16	(stall with pipeline) and L15	33
☐	L15	tremblay.in. and sun.as.	99
☐	L14	(stall with pipeline) and L13	3
☐	L13	(murty).in. and intel.as.	3
☐	L12	(murty and shaw).in. and intel.as.	0
☐	L11	5860000.uref.	4
☐	L10	6279100.uref.	6
☐	L9	204535.ap. and sun.asn.	1
	DB=USPT; PLUR=NO; OP=ADJ		
☐	L8	6044456.pn.	1
☐	L7	5860000.pn.	1
☐	L6	6012139.pn.	1
☐	L5	6012139.pn.	1
☐	L4	5860000.pn.	1
	DB=PGPB,USPT; PLUR=NO; OP=ADJ		
☐	L3	6044456.uref.	2
☐	L2	604456.uref.	0
☐	L1	intel.asn. and murty.in.	3

END OF SEARCH HISTORY